# Evaluation of Computer Vision Models on Car Crash Detection

**Navaneeth Dontuboyina**

*Received September 20, 2023*
*Accepted February 07, 2024*
*Electronic access February 15, 2024*

This paper focuses on the Car Crash Dataset (CCD) which consists of dashcam footage capturing car crashes and non-crash scenarios. The dataset is derived from the Berkeley Deep Drive dataset (BDD100K) and includes annotations indicating crash presence in each frame. Specifically, the study emphasizes videos in which the accident is recorded from the driver's point of view, resulting in 801 videos with such involvement. First step was to organize and segment the videos, and subsequently filter them to retain footage filmed from the car's point view (or known in the dataset as ego-vehicles). Differing from previous studies that also tried to predict crashes between other, non-ego vehicles, this study is only focused on crashes between ego-vehicles and other objects because it is a logical first step for any autonomous vehicle to immediately stop if it senses it will crash. From there, research can be expanded to include crashes between other objects. This study does not use bounding boxes or other methods to isolate different components or objects within the driving footage. There are two approaches for modeling. The first employed pre-trained Long Short-Term Memory (LSTM) models such as InceptionV3, Visual Geometry Group 16 (VGG16), Residual Network 50 (ResNet50), chosen for their efficiency in image datasets and computer vision. The dataset was refined to exclude videos lacking ego-vehicle involvement. The data was then split for training and testing, and label processing was performed. Each model underwent feature extraction and utilized a Recurrent Neural Network (RNN) sequence model for sequential prediction. The second approach entailed a custom Convolutional Neural Network (CNN) model, differing from the LSTM framework. Results were documented in comma separated values (csv) files, and 40 non-crash observations were randomly selected to balance the pre-existing 40 crash observations. Threshold analysis was conducted using receiver operating characteristic (ROC) curves, determining optimal values for crash classification. The InceptionV3 model achieved the highest accuracy at 93%, with 2 false positives and 3 false negatives. VGG16 and ResNet50 achieved 87% and 88% accuracy, respectively, with 3 false positives and 7 false negatives. The custom model proposed in this research paper has exhibited 64% accuracy, with 19 false positives and 10 false negatives. The findings indicate promise in crash prediction for autonomous vehicles, with pretrained models showing robust performance. However, the custom model displayed lower accuracy and higher false negatives. There are, obviously, limitations to this paper that include a very general custom model, not much depth into the use of V2X technologies for car crash prediction and the use of only one camera angle as opposed to multiple angles from the car's perspective. All of these improvements are being discussed for further research in a follow-up paper. This study introduces a practical approach for autonomous vehicles to identify the primary threat of a front-end collision without relying on CCTV or similar footage, as seen in previous research. It also delves into how and if pre-trained LSTM models can be integrated into the complex architecture of computer vision models in the future.

## Introduction

The purpose of this research paper is to see whether or not simple, pre-trained LSTM keras models could sufficiently predict car crash detection and thus could be used as a part of complex CV model architecture. There have already been projects and studies on using computer vision to predict potential car crashes and aid with autonomous vehicles. Two projects/studies conducted by the same team of Dhananjai Chand, Savyasachi Gupta and Goutham K. at National Institute of Technology (NIT) Warangal, India give insight into this with two different types of ego-vehicle footage [1,2]. "Computer Vision-based Accident Detection in Traffic Surveillance" uses closed-circuit television (CCTV) footage from a variety of lighting and weather conditions in high urban intersections in India. The framework for the machine learning algorithm utilizes a Mask R-CNN [3] which is used specifically for its region of interests (RoIs) align feature. This model also uses a box offset regressor, softmax classifier and a mask fully convolutional network (FCN) predictor. They classify the cars using bounding boxes and then try to predict around 15 frames ahead the likelihood of the bounding boxes hitting each other by assessing their vector angles and the acceleration of these objects. This model has a 71% detection rate and a 0.53% false alarm rate. "Computer Vision Based on Accident Detection for Autonomous Vehicles" is more related to this research topic as it captures the similar type of data which is crash footage from the dashcam and tries to achieve the similar sort of application for a self-driving car. It uses roughly a similar model as before with the same technique of drawing binding boxes and calculating the likelihood of a crash a few frames

© The National High School Journal of Science 2023

ahead. However, it tries to predict the crashes between other cars and other objects, not the ego-vehicle. It has a 79% accident rate but with a 34% false alarm rate. The VSLab paper titled "Anticipating Accidents in Dashcam Videos" uses dashcam footage in high population areas all across Taiwan[4]. However, it is not predictive but rather labels each video as a crash or no crash based on the footage leading up to it. My project is similar to the one in "Computer Vision Based on Accident Detection for Autonomous Vehicles" in style but different in the intent. This project will also utilize dashcam footage, however this project will only try to identify crashes that pose an immediate threat to the ego-vehicle rather than identifying surrounding crashes. This paper can contribute to the development of AV technology which will in turn contribute to safer, fuel-efficient, and faster transportation for society in the private sector within industries like ride-sharing as well as in public transportation. This research will also make transportation more equitable because pushing the advancement of AVs benefits those whose mobility is hindered due to medical or psychological reasons (individuals incapable of driving)[5].

```
def make_dataset():
    data_new = pd.read_csv('Crash_Table.csv')
    row, col = data_new.shape

    for index, row in data_new.iterrows():
        if row["egoinvolve"] == "No":
            data_new = data_new.drop(index)
    data_new = data_new.drop("startframe", axis='columns')
    data_new = data_new.drop("youtubeID", axis='columns')
    data_new = data_new.drop("timing", axis='columns')
    data_new = data_new.drop("weather", axis='columns')
    data_new = data_new.drop("egoinvolve", axis='columns')

    names = os.listdir("./normalframes")
    header = list(data_new.columns)
    data_normal = pd.DataFrame(columns=header)

    for i in names:
        data_normal.loc[len(data_normal.index)] = [i, 0, 0, 0, 0, 0, 0, 0, 0,

    data_new = pd.concat([data_normal, data_new], axis=0)
    data_new.drop(data_new.iloc[:, 1:50], inplace=True, axis=1)

    # Using DataFrame.sample()
    train_df=data_new.sample(frac=0.8,random_state=200)
    test_df=data_new.drop(train_df.index)

    return [train_df, test_df]

list1 = make_dataset()
train_df = list1[0]
test_df = list1[1]
```

**Fig. 1** Formatting the CSV to only include the 50 frames and whether there was a crash at the end. Also separating a certain proportion of them into a train and test data frame.

## Methodology

The Car Crash Dataset (CCD)[6] is a Kaggle dataset that is compiled from a GitHub dataset[7]; for our purposes we will just use the original on GitHub. This original dataset is taken from a random sample of dashcam footage of car crashes and non-car crashes from the BDD100K. It is an image-based data in the form of .mp4 files. The GitHub dataset contains a Crash1500 folder, Normal folder, and a text file that contains useful information and labels. For each of the 50 frames of each video, the text file has a 0 or a 1 on whether a crash is present within that frame or not. There are also other helpful labels for what time of day, the climate, and whether the ego-vehicle is involved or not. The videos from the Crash1500 folder which involve the ego-vehicle are the ones that we are concerned with. There are 801 of these such videos and 3000 normal videos. Using a basic python script, the dataset was downloaded and two new folders for the crash and non-crash videos were made. Then each video was split up into 50 frames and stored in its own folder. The newly made crash folders were then filtered to only include videos with ego-vehicles involved. These frames can now be fed into the models.

For the models, there are three different variations of one method as well as an entirely different method to compare results. The first way is to use the pre-trained LSTM models on Keras - specifically the InceptionV3, VGG16 and ResNet50 models because they have the best prediction accuracy on image datasets and for computer vision according to the Keras website. Next is to filter out the downloaded dataset by iterating through the crash csv downloaded from the original Kaggle dataset and then dropping videos that do not have an ego-car crash involved within them. Then to add the information of non-crash videos to this csv to create one big documentation. Next, perform a train test split on a csv-to-pandas dataframe of all the videos with the video name and the added variable of 1 (crash) or 0 (non-crash).

Next, perform a label processing stage which is independent from the train test split simply because there are only two labels for any video. A feature extractor must be built for each of the different models and used on the video within the train and test. Then for each of the models there is an RNN that is specifically used for the sequential prediction task of this project and to train my entire model with the train dataset. Inferences can now be made on my test dataset.

For the second method a custom CNN model was built rather than using a Keras LSTM - every other step stayed the same. The CNN model consisted of six pairs of 16 input 3x3 convolutional layers and then a 2x2 max-pooling layer with rectified linear unit (ReLU) activation. This is then flattened and finished with a softmax activation layer to give a percentage of how likely a crash or non-crash occurs. The architecture for this custom model was inspired by the combined architectures of all the previous model's CNN components. The point of this custom model is to see how simple and basic a model would have to be in order to predict car crashes. For each model, the results were uploaded to its own csv file for documentation and then 40 non-crash observations were randomly selected to match the 40 crash observations. I analyzed these results by finding the most optimal threshold value for each model to classify as


© The National High School Journal of Science 2023

```
path = "/Users/nalthan/Desktop/vertiasopencvprojecy/Crash-1500"

crash_file_names = [path + "/" + filename for filename in os.listdir(path) if filename.endswith('.mp4')]
print(crash_file_names)
for vid in crash_file_names:
    cam = cv2.VideoCapture(vid)
    currentframe = 1
    tempdir = "/Users/nalthan/Desktop/vertiasopencvprojecy/crashframes/"
    crashframedir = vid.replace("Crash-1500", "crashframes").replace(".mp4", '')
    os.mkdir(crashframedir)
    while(True):
        # reading from frame
        ret,frame = cam.read()

        if ret:
            # if video is still left continue creating images
            name = crashframedir + "/frame" + str(currentframe) + '.jpg'
            print ('Creating...' + name)

            # writing the extracted images
            cv2.imwrite(name, frame)

            # increasing counter so that it will
            # show how many frames are created
            currentframe += 1
        else:
            break
    cam.release()
    cv2.destroyAllWindows()
```

**Fig. 2** Creating individuals frames for each video which has a car crash in it

```
path = "/Users/nalthan/Desktop/vertiasopencvprojecy/Normal"

normal_file_names = [path + "/" + filename for filename in os.listdir(path) if filename.endswith('.mp4')]
print(normal_file_names)
for vid in normal_file_names:
    cam = cv2.VideoCapture(vid)
    currentframe = 1
    tempdir = "/Users/nalthan/Desktop/vertiasopencvprojecy/normalframes/"
    normalframedir = vid.replace("Normal", "normalframes").replace(".mp4", '')
    os.mkdir(normalframedir)
    while(True):
        # reading from frame
        ret,frame = cam.read()

        if ret:
            # if video is still left continue creating images
            name = normalframedir + "/frame" + str(currentframe) + '.jpg'
            print ('Creating...' + name)

            # writing the extracted images
            cv2.imwrite(name, frame)

            # increasing counter so that it will
            # show how many frames are created
            currentframe += 1
        else:
            break
    cam.release()
    cv2.destroyAllWindows()
```

**Fig. 3** Creating individuals frames for each video which does not have a car crash in it

```python
def build_feature_extractor():
    feature_extractor = tf.keras.applications.ResNet50(
        include_top=True,
        weights="imagenet",
        input_tensor=None,
        input_shape=None,
        pooling=None,
        classes=1000,

    )



    preprocess_input = tf.keras.applications.resnet.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```

**Fig. 4** Sample feature extractor for ResNet50 model

```python
# For each video.
for idx, path in enumerate(video_paths):
    print(idx, path)
    print(type(path))
    # Gather all its frames and add a batch dimension.
    frames = load_video(os.path.join(root_dir, str(path).zfill(6)+'.mp4'))
    frames = frames[None, ...]

    # Initialize placeholders to store the masks and features of the current video.
    temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
    temp_frame_features = np.zeros(
        shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )

    # Extract features from the frames of the current video.
    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            temp_frame_features[i, j, :] = feature_extractor.predict(
                batch[None, j, :]
            )
        temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    frame_features[idx,] = temp_frame_features.squeeze()
    frame_masks[idx,] = temp_frame_mask.squeeze()

return (frame_features, frame_masks), labels


train_data, train_labels = prepare_all_videos(train_df, "/Users/nalthan/Desktop/vertiasopencvprojecy/train")
test_data, test_labels = prepare_all_videos(test_df, "/Users/nalthan/Desktop/vertiasopencvprojecy/test")
```

**Fig. 5** ResNet50 feature extractor being applied to the the train and test data

a crash or non-crash using the ROC curves from sklearn kit python library. After determining the threshold value that yields the most accuracy on the test data for each of the models, a confusion matrix was made to see how well the model predicted for the test data.

## Results

The InceptionV3 model achieved 92% accuracy while reporting two false positives and three false negatives of whether a car crash occurs. The VGG16 model and ResNet50 model achieved 87% and 88% accuracy respectively while reporting three false positives and seven false negatives of whether a car crash occurs. The custom model developed for this research achieved 64% accuracy while reporting 19 false positives and 10 false negatives of whether a car crash occurs.

| | Thresholds | Accuracy |
|---|---|---|
| **8** | 0.2958 | 0.938272 |
| **10** | 0.2827 | 0.925926 |
| **9** | 0.2879 | 0.913580 |
| **12** | 0.2343 | 0.876543 |
| **13** | 0.2246 | 0.876543 |

**Table 1** The top 5 threshold values and their corresponding accuracy rates for the InceptionV3 model

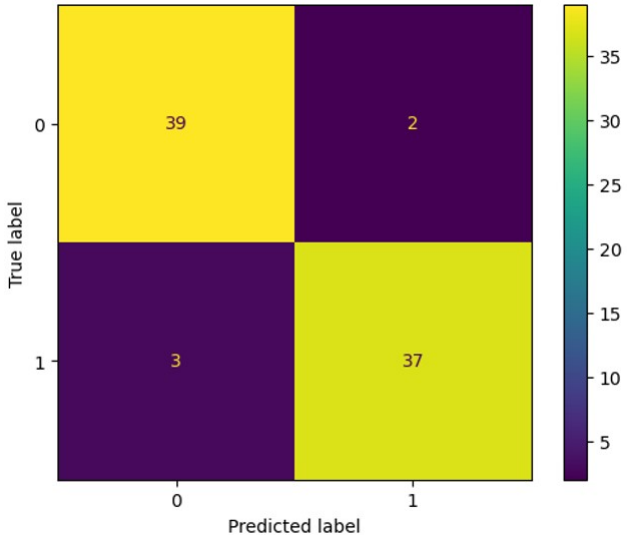


**Fig. 6** Confusion Matrix for Inception V3 model

| | Thresholds | Accuracy |
|---|---|---|
| **8** | 0.2370 | 0.876543 |
| **9** | 0.2289 | 0.876543 |
| **6** | 0.2471 | 0.876543 |
| **7** | 0.2421 | 0.864198 |
| **4** | 0.3011 | 0.839506 |

**Table 2** The top 5 threshold values and their corresponding accuracy rates for the VGG16 model

## Discussion

These models have comparable results to previous, aforementioned studies and projects in predicting and identifying car


© The National High School Journal of Science 2023

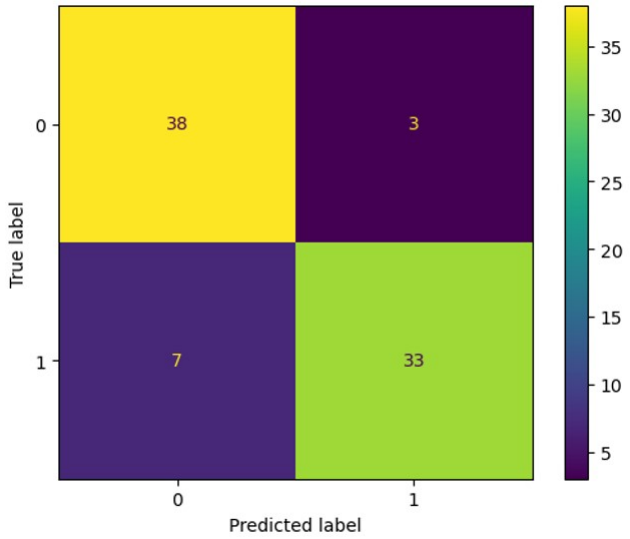


**Fig. 7** Confusion Matrix for VGG16 model

| | Thresholds | Accuracy |
|---|---|---|
| **8** | 0.1874 | 0.888889 |
| **6** | 0.1952 | 0.876543 |
| **7** | 0.1917 | 0.864198 |
| **10** | 0.1825 | 0.839506 |
| **9** | 0.1831 | 0.827160 |

**Table 3** The top 5 threshold values and their corresponding accuracy rates for the ResNet50 model

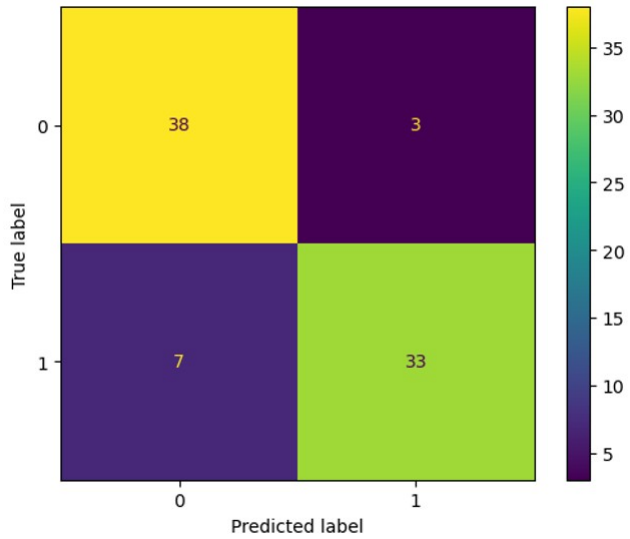


**Fig. 8** Confusion Matrix for ResNet50 model

| | Thresholds | Accuracy |
|---|---|---|
| **23** | 0.2163 | 0.641975 |
| **12** | 0.2433 | 0.641975 |
| **16** | 0.2292 | 0.629630 |
| **25** | 0.2149 | 0.629630 |
| **24** | 0.2157 | 0.629630 |

**Table 4** The top 5 threshold values and their corresponding accuracy rates for the custom model

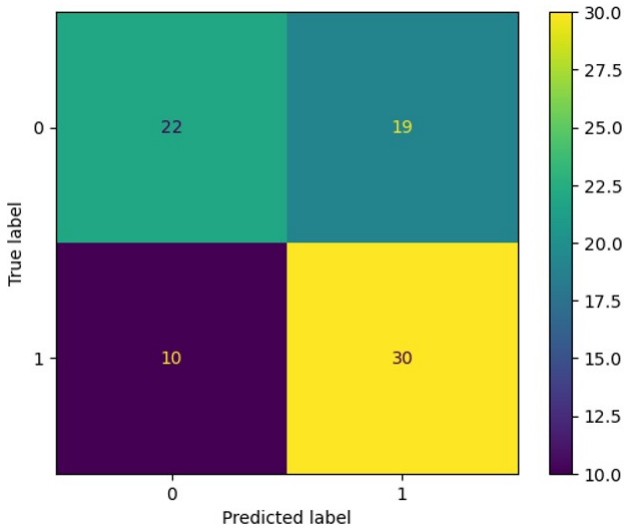


**Fig. 9** Confusion Matrix for the custom model

crashes for autonomous vehicles. This study's accuracy rates of 93.8%, 87.6%, 88.8% and 64.1% for the LSTM and CNN custom model respectively are similar, if not better, to the NIT Warangal prediction rate of 71% and the VSLab rate of 79%. These results are probably because the past studies tried to use more advanced and complex methods such as bounding boxes and individual object detection as well as trying to account for more factors like crashes that do not include the self-car. The more experimental nature of these past studies result in lower prediction rate even though the methods are quite fascinating and innovative. However, it is important to note that the custom built model performed worse in terms of accuracy and minimizing false negatives. The reason why the InceptionV3 model did the best is most likely because it possesses the fastest GPU and CPU inference time of 42.2 ms and 6.9 ms respectively. This study presents an affirmative step into how an autonomous vehicle would realistically identify the most basic threat of a crash from the front rather than utilize CCTV or other types of footage as used in previous research. It also suggests that the process of identifying these basic threats is best when using a pre-trained computer vision model and combining that with

an RNN structure. These results mean that in the future when industries are developing CV models for car crash prediction and detection, they are predicted to use a simple LSTM that is most like the architecture of the InceptionV3 model due to its fast inference time and its formidable accuracy rate. Beyond autonomous vehicles, there are far more implications in other parts of the automotive industry based on this study. Potentially, traffic signals and city-wide traffic management systems can use these CV models to help in flow of traffic, or identify traffic violators. Insurance companies can implement CV models to identify who is at fault in an accident.

## Conclusions

The research has led to a conclusion that the InceptionV3 model achieved the highest accuracy at 93%, with 2 false positives and 3 false negatives. VGG16 and ResNet50 achieved 87% and 88% accuracy, respectively, with 3 false positives and 7 false negatives. The custom model proposed in this research paper has exhibited 64% accuracy, with 19 false positives and 10 false negatives. The findings indicate promise in crash prediction for autonomous vehicles, with pretrained models showing robust performance. However, the custom model displayed lower accuracy and higher false negatives. For further research, the next step is to most certainly incorporate real time decision making. A model that can take in a video and predict a certain amount of frames or seconds ahead as to whether or not a crash will occur lends itself more useful to self-driving cars. Then another addition would be to do more research into specifically building a custom model for predicting car crashes. The custom model used in this research was basic and more general which is why the accuracy paled in comparison to the pretrained models. Increasing the model to handle longer video lengths and inputs for videos from different angles of the car such as the back, side and blindspots for the same occurrence will be an improvement to look out for as this is how car manufacturers collect data for their autonomous vehicles. Another next step I would like to pursue to research more to improve predictability is on autonomous cars with Connected Vehicle (CV) technologies such as vehicle-to-everything (V2X) - vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) and vehicle-to-pedestrian (V2P)[8]. These connected vehicle technologies would provide communication among vehicles in their nearby area and their surrounding environment. For a crash detection and prediction computer vision model to be more accurate, it must also take into account CV technology and their different V2X facets as different inputs.

## Acknowledgements

Thank you for the guidance of Mr. Nowell Closser from Harvard University in the development of this research paper.

## References

1 P. Ijjina, D. Chand, S. Gupta and K. Goutham, 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT, Kanpur, India, pp. 1–6,.

2 S. Chand and I. Kavati, 2020 IEEE 17th India Council International Conference (INDICON, New Delhi, India, pp. 1–6,.

3 Kaiming, *Mask R-CNN*, arXiv.Org, 24 Jan. 2018, arxiv.org/abs/1703.06870.

4 Wentao, Proceedings of the 28th ACM International Conference on Multimedia.

5 *of Self-Driving Vehicles." Coalition For Future Mobility, coalitionforfuturemobility.com/benefits-of-self-driving-vehicles/*, Accessed 1 Feb.

6 A. Jamil, *Kaggle*, – – –.

7 Wentao, *GitHub*, 2012.

8 *U.S. Department of Transportation*, – – – – –.


© The National High School Journal of Science 2023